Native mesh ordering with Scotch 4.0

François Pellegrini

INRIA Futurs – Project ScAlApplix
pelegrin@labri.fr

Abstract. Sparse matrix reordering is a key issue for the the efficient factorization of sparse symmetric matrices, not only to reduce fill-in and operation count, but also to increase concurrency in the elimination tree, which is essential to achieve high speed-ups when solving the systems on parallel architectures. Most sparse matrix reordering software, when applied to finite-element or finite-volume meshes, require the construction of a nodal adjacency graph, the size of which is linear in the number of elements of the meshes, but quadratic in the number of nodes per element, resulting in a memory bottleneck for large 3D meshes of brick elements. This paper presents the native mesh ordering capabilities of the Scotch 4.0 software package, which has been designed so as to compute efficient orderings of mesh structures in a time which is still asymptotically quadratic function of number of nodes per element, but linear in space with respect to the number of elements and nodes, allowing for the processing of larger meshes on classical workstations.

Keywords: sparse matrix reordering, mesh partitioning, graph partitioning.

1 Introduction

When solving large sparse linear systems of the form $Ax = b$, it is common to precede the numerical factorization by a symmetric reordering. The aim of the reordering procedure is to produce a permutation matrix P, such that pivoting down the diagonal of the resulting permuted matrix PAP^T produces substantially lower amount of fill-in and requires less arithmetic operations than pivoting down the diagonal of the original matrix A.

The two most classically used reordering methods are Minimum Degree and Nested Dissection. The Minimum Degree algorithm [4, 8] is a local heuristic that performs its pivot selection by picking from the graph, one after another, nodes of minimum degree. Although this algorithm is fast, it is intrinsically sequential, and very little can be theoretically proven about its efficiency. On the other hand, many theoretical results have been carried out on Nested Dissection ordering [7], and its divide and conquer nature makes it easily parallelizable. In practice, Nested Dissection produces orderings which, both in terms of fill-in and operation count, compare well to the ones obtained with Minimum Degree [6, 9]. Moreover, the elimination trees induced by nested dissection are

broader, shorter, and better balanced, and therefore exhibit much more concurrency in the context of parallel Cholesky factorization. Therefore, many sparse matrix reordering software combine the strengths of the two classes of methods by performing Nested Dissction at first, and then turn to Minimum Degree methods to order smaller subgraphs that can be handled on a single processor, such that the unbalanced trees computed by the Minimum Degree algorithms cannot harm concurrency [5, 10].

Most software which compute nested dissection orderings [6, 9] handle adjacency matrices as adjacency graphs, the nodes of which represent the variables, and such that there exists an edge between two vertices if there exists a non-zero off-diagonal coefficient between these two variables. Such graph reordering software have proven very efficient, as their sequential space and time complexities are almost linear with respect to the number of edges of the adjacency graphs they handle. However, if the time complexity is usually not a problem in itself, the space complexity can become a serious bottleneck for large 3D graphs. Indeed, when reordering the nodes of a mesh with such software, and in order to prevent any two nodes of the same element to belong to different parts, one has to build a nodal adjacency graph such that any two nodes that have an element in common are connected by an edge, resulting in a number of edges that is linear in the total number of nodes but quadratic in the number of nodes owned by the elements. If this is not too important for 2D or tetrahedral meshes, this may lead to a serious bottleneck for large 3D meshes made of brick elements.

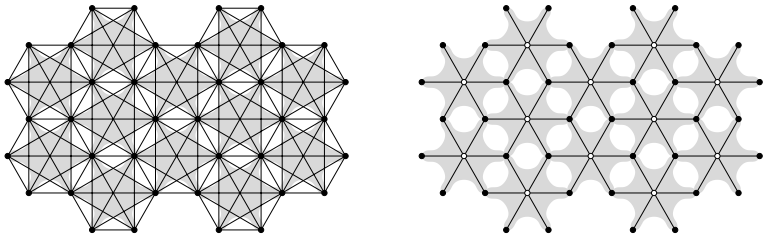
While the time complexity cannot be reduced, as most of the algorithms require the traversal of all of the adjacency lists, it is yet possible to address the memory bottleneck by providing algorithms that operate directly on the mesh structure, which is only linear in the number of nodes per element.

This paper presents the native mesh ordering capabilities of Scotch 4.0, a software package for graph and mesh partitioning. It is structured as follows: section 2 introduces the data structure used to represent meshes; section 3 outlines the algorithms that are used in our nested dissection based mesh ordering, with respect to their graph counterparts; section 4 presents some efficiency results. Final section gives the conclusion.

2 Mesh structure

Most of the basic algorithms which are used to compute the vertex separations and build the submeshes require the adjacency lists for every node and every element. This requires the mesh structure to be defined as a symmetric bipartite graph, where the nodes are only connected to the elements and vice versa. For instance, Fig. 1 shows a set of hexagonal elements the nodes of which have to be reordered, which is represented both as a nodal graph, with all nodes of every element being connected into cliques, and as a mesh bipartite graph.

The memory savings that the mesh structure can yield compared to the graph structure can be easily evaluated in the case of regular meshes. Let us consider a regular mesh with e elements and n nodes, such that on average every element



a. Graph. **b.** Mesh.

Fig. 1. Nodal and bipartite mesh graphs representing the same set of hexagonal elements. Black vertices represent the nodes to reorder, and white vertices represent the elements

is connected to ζ nodes, and every node is connected to η elements. Since the overall number of arcs that are incident to the elements of the mesh is equal to the overall number of arcs that are incident to the nodes, $\zeta e = \eta n$ holds. The mesh structure has ηn edges, and the size of its index data structure is in $\Theta(e+n)$, so the overall memory occupation of the mesh, in terms of symmetric arcs, is in $\Theta(n(\eta/\zeta+2\eta+1))$. The corresponding nodal graph has only n nodes, δ node neighbors on average, and the size of its index data structure is in $\Theta(n)$, so the overall memory occupation of the graph is in $\Theta\left(n(\delta+1)\right)$. As shown in Table 1, the mesh structure should only be used when the ratio of nodes per element is high enough; else, the graph structure should be preferred.

Table 1. Memory occupation of the graph and mesh structures with respect to the number of nodes for several classical mesh topologies

Mesh type	δ	ζ	η	Graph size	Mesh size	Ratio
Triangles	6	3	6	$7n$	$15n$	2.14
2D elements, 8 points	8	4	4	$9n$	$10n$	1.11
3D elements, 26 points	26	8	8	$27n$	$17n$	0.62
Hexagons (e.g. HROT elements)	12	6	3	$13n$	$\frac{15}{2}n$	0.57

3 Mesh algorithms

As for graphs, the mesh-based nested dissection framework comprises two basic blocks: the computation of the vertex separators, and the building of the induced submeshes for the separator and for the two separated parts.

3.1 Computation of vertex separators

The mesh separation routine is based on the same multi-level framework as the one that has been succesfully used in the context of graph separation [2, 6]. The original mesh to partition is coarsened into a set of smaller meshes until some size threshold is reached, the smallest mesh is partitioned using a fast greedy algorithm, and the partition is successively projected back to the larger meshes. In order to preserve the quality of the partition during the uncoarsening phase, the projected partition is refined after each uncoarsening step by using a mesh version of a Fiduccia-Mattheyses [3] local optimization algorithm.

The use of mesh structures rather than of graphs required the definition of mesh-oriented versions of the separator computation and optimization algorithms. Instead of moving individual nodes from one of the parts to the separator and vice versa, mesh algorithms move entire elements from one part to another. Nodes that have all of their neighboring elements in the same part are said to belong to this part, or else belong to the separator.

Our mesh-oriented Fiduccia-Mattheyses algorithm is based on a dynamic gain structure which records, for all of the frontier elements (that is, the elements that have neighboring nodes in the separator), the possibly negative increase of the size of the separator, as well as the improvement of the balance between the two parts, that swapping the element would yield. Once an element of best gain is chosen and swapped, the parts to which its neighboring nodes belong are updated, as well as the gains of the elements that neighbor these neighboring nodes.

3.2 Mesh coarsening

Like for graphs, the quality of mesh coarsening is essential to the computation of efficient vertex separators. Several algorithms have been tested to date. The first one collapses elements by collapsing them into single nodes; this algorithm dramatically reduces the number of nodes at each stage, but tends to modify the topological structure of the original mesh, resulting in poorer partitions. The second one matches pairs of nodes that belong to the same elements, much similarly to how traditional graph coarsening algorithms perform on nodal graphs. The third one, which has been the most satisfactory to date, and is used in our numerical evaluations, merges pairs of elements that have the largest number of nodes in common. After elements are merged, nodes that have the same adjacency pattern are collapsed into nodes of larger weights, so that the number of nodes decreases along with the number of elements. This collapsing process is similar in nature to the one of graph compression [1], but occurs after each coarsening step, and not just once.

3.3 Computation of induced submeshes

While, for graph ordering, the computation of the subgraphs induced by the separator and by each of the two parts is very similar in nature, the computation

of induced submeshes differs quite for separators and for parts. For mesh parts, the algorithms is very similar to its graph counterpart: the nodes that belong to the selected part are flagged, as well as all of their neighboring elements, and all of these selected vertices are used to create a new submesh that will be used for subsequent Nested Dissection. Un-flagged nodes of flagged elements are also kept in every submesh as halo nodes, that is, nodes that will not be considered for further reordering in this part, but the adjacency of which will be accounted for when computing the degree of the submesh vertices in our mesh version of the Halo Approximate Minimum Degree algorithm [10].

The separator mesh building algorithm is more complex. As a matter of fact, when building the minimal set of elements that connect the selected separator nodes, it is very common to find elements that belong to different parts but yet have the same adjacency information with respect to the separator nodes. Duplicate elements that would bear no additional connectivity information are therefore identified and discarded during the contruction of the separator submesh, so that subsequent ordering algorithms perform more efficiently.

4 Performance evaluations

To validate our approach, we have run the graph and mesh versions of Scotch 4.0 on the meshes and graphs described in Table 2, which come either from an industrial parallel electromagnetics code used by CEA/CESTA, or from the Parasol benchmark. The order of the systems is equal to their number of nodes.

Table 2. Test meshes and nodal graphs

Name	Description	Nodes	Elements	Mesh edges	Graph edges	ζ	η
altr4	3D, HROT	26089	16638	99828	163038	6.00	3.82
chanel1m	3D, HROT	80592	54899	329394	526860	6.00	4.08
conesphere	3D, HROT	1055039	887428	5324568	8023236	6.00	5.04

The results presented in table 3 have been achieved on a 375MHz Power3-based RS6000 machine with 8 Gb of main memory. They represent the amount of memory and CPU time required to perform nested dissection up to parts of approximately one hundred nodes, and then run Patrick Amestoy's mesh version of the Halo Approximate Minimum Degree algorithm on the resulting submeshes.

These results are consistent with our model. The use in the mesh version of two nested loops and of hash tables to iterate on all of the neighbor elements of some element is indeed more expensive than just reading a plain list of neighbors but, as mesh sizes increase, increased data locality compensates for it, allowing to run real 3D test cases of more than two million nodes on a high-end workstation.

Table 3. Performance results for the graph and mesh versions of SCOTCH 4.0. Memory spendings are in kb and times are in CPU seconds

Name	Graph				Mesh			
	NNZ	OPC	Memory	Time	NNZ	OPC	Memory	Time
altr4	2.03e+06	3.93e+08	10673	1.7	1.94e+06	3.86e+08	11268	4.0
chanel1m	8.94e+06	2.62e+09	36869	6.4	8.80e+06	2.72e+09	20050	15.8
conesphere	5.58e+08	1.80e+12	155883	122.2	6.01e+08	2.06e+12	52125	402.2

5 Conclusion

On-going work regards the study of more efficient mesh coarsening algorithms, which are critical for performance, both in terms of quality and time.

The LIBSCOTCH 4.0 library is available, under the LGPL license, from the SCOTCH web page, at `http://www.labri.fr/~pelegrin/scotch`. SCOTCH can therefore be used as a testbed for the creation and evaluation of new partitioning and ordering techniques, for which contributors are welcome.

References

[1] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Comput.*, 16(6):1404–1411, 1995.

[2] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Autom. Conf.*, pages 175–181. IEEE, 1982.

[4] A. George and J. W.-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[5] P. Hénon, P. Ramet, and J. Roman. PaStiX: A high-performance parallel direct solver for sparse symmetric definite systems. *Parallel Computing*, 28(2):301–321, January 2002.

[6] G. Karypis and V. Kumar. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0*. University of Minnesota, September 1998.

[7] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal of Numerical Analysis*, 16(2):346–358, April 1979.

[8] J. W.-H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11(2):141–153, 1985.

[9] F. Pellegrini and J. Roman. Sparse matrix ordering with SCOTCH. In *Proceedings of HPCN'97, Vienna, LNCS 1225*, pages 370–378, April 1997.

[10] F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Practice and Experience*, 12:69–84, 2000.